Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Evaluating Global Log File Analysis Systems: Design Frameworks Using Ensemble Techniques

Rahul B. Pawar*1, Dr. Rajesh K. Shukla*2

*1 Research Scholar Computer Science and Engineering, Oriental University, Indore (M.P), India

rahulpwr023@gmail.com

*2Supervisor, Department of Computer Science and Engineering, Oriental University, Indore (M.P), India

rajeshshukla@oriental.ac.in

ARTICLE INFO

Received: 02 Nov 2024

Revised: 25 Dec 2024

Accepted: 26 Dec 2024

ABSTRACT

Log file analysis plays a vital role in performance optimization, security monitoring, and fault detection across large-scale international networks. However, traditional log analyzers struggle to process modern log data due to its high volume, diverse formats, and real-time generation. This study proposes a generic log analysis system using distributed computing to improve scalability and efficiency. The methodology involves collecting multiple log types, including firewall, server, web, email, and call data logs, and processing them using Apache Hadoop MapReduce for large-scale batch analysis. Log events are parsed, aggregated, and summarized to identify patterns, abnormal activities, and usage trends without interfering with system performance. Experimental results show that the proposed approach successfully analyzes diverse log formats and produces meaningful summaries while reducing manual effort. The system demonstrates improved handling of large log datasets and supports visualization for better interpretation. Overall, the proposed log analyzer provides an efficient and scalable solution for managing and extracting insights from heterogeneous log data.

Keywords: Log file analysis, Hadoop, Spark, Flink, big data, distributed computing, real-time processing.

Introduction

Modern digital infrastructures generate enormous volumes of log data that capture system events, user activities, errors, performance metrics, and security incidents. These logs originate from heterogeneous environments such as cloud platforms, web applications, network devices, industrial control systems, and hospital information systems. As organizations expand globally, log streams become increasingly complex due to diverse formats, distributed deployment, varying time zones, and continuously evolving system behaviors. Consequently, traditional log analysis tools which typically rely on manual inspection or single-machine processing are insufficient for extracting timely and reliable insights from large-scale global log repositories.

Global log file analysis is essential for several key objectives: ensuring system availability, identifying operational bottlenecks, detecting intrusions, supporting compliance auditing, and enabling proactive maintenance. However, achieving these goals in real time requires scalable processing frameworks capable of handling high throughput, low latency, and fault tolerance. Big data technologies such as Hadoop, Spark, and Apache Flink have emerged as effective solutions for batch and stream-based log processing. Among these, Apache Flink is particularly suitable for real-time analytics due to its event-

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

time processing, windowing operations, and stateful computation capabilities, which allow accurate analysis even under out-of-order event arrival.

Beyond scalable processing, the challenge of accurate anomaly detection and intelligent classification remains significant. Log data often contains noise, missing fields, irregular patterns, and domain-specific variations, making single-model prediction approaches unreliable in many real-world conditions. To address this limitation, ensemble learning techniques provide a robust solution by combining multiple predictive models to improve generalization and reduce false alarms. Ensembles such as bagging, boosting, and stacking leverage model diversity to better capture complex system behaviors and rare incident patterns, especially in cybersecurity and fault detection scenarios.

This study evaluates existing global log file analysis systems and proposes a design framework that integrates distributed processing with ensemble-based intelligence. The framework supports heterogeneous log formats, scalable ingestion, automated preprocessing, and enhanced anomaly detection, ultimately enabling organizations to make faster, more informed operational decisions across industries, healthcare, and large-scale network environments.

1. Importance of Log Files in Modern Systems : Log files record system events, errors, transactions, and security-related incidents, making them essential for monitoring and understanding system behavior. As digital infrastructure expands rapidly, organizations increasingly depend on reliable log analysis solutions to detect failures, ensure security, and optimize overall performance.

2. Growth of Big Data and Need for Advanced Analytics : Today, massive volumes of digital data are generated in critical domains such as social networks, e-commerce, healthcare, education, and environmental systems. This rapid growth has made **big data mining** an important technique for extracting valuable insights that support decision-making and enable more personalized services. As a result, many advanced computing frameworks for large-scale data analysis have been developed.

3. Adoption of Big Data Frameworks for Log Analysis : Traditional log analysis tools cannot handle large-scale, heterogeneous, and continuously generated logs in real time. To overcome this limitation, big data technologies such as **Apache Hadoop, Apache Spark, and Apache Flink** are widely used for scalable and efficient log processing.

4. Why Apache Flink is Suitable for Log Analytics : Apache Flink is a mature, open-source distributed stream processing framework designed for large-scale analytics. It supports both **bounded data (batch logs)** and **unbounded data streams (real-time logs)**, enabling low-latency and high-throughput processing. Logs contain structured entries describing system activities and incidents, including security-related events, making Flink highly suitable for continuous monitoring and anomaly detection.

5. Anomaly Detection in Distributed Logs Using Flink : This research focuses on identifying anomalies in large-scale logs collected from distributed environments using Flink. Flink ensures reliability through fault tolerance mechanisms such as:

- **Periodic checkpointing**, where application state is stored persistently for automatic recovery during failures.
- **Savepoints**, which store consistent execution states to allow stopping, resuming, upgrading, or restarting Flink jobs without losing progress.
  Both checkpointing and savepoints work asynchronously, enabling continuous processing without interrupting the flow of incoming log data.

6. Motivation for a Generic Multi-Log Analyzer: Many existing log analyzers in the market can process only one specific type of log file. Therefore, the main objective of this system is to develop a **generic log analyzer** capable of supporting multiple log formats and improving productivity through

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Flink-based distributed processing. By using Apache Flink, the system becomes more efficient in processing diverse log datasets.

7. Integration with Kafka for Real-Time Streaming: In the proposed framework, logs are collected and streamed through Kafka and then processed using Apache Flink. This integration allows continuous data ingestion, real-time orchestration, and generation of meaningful insights. Performance evaluation also confirms that the combined Flink–Kafka approach enhances scalability and supports real-world use cases effectively.

8. Role of Machine Learning and Ensemble Techniques: To further improve accuracy and robustness, ensemble learning techniques can be applied to log analysis. By combining multiple predictive models trained using different methods or subsets of data, ensemble approaches improve anomaly detection and generalization compared to single-model methods. This makes ensemble learning valuable for large-scale and complex log environments.

Log File Analysis Overview (Step-by-Step)

The overall log analysis process follows these main steps:

1. **Data Collection:** Collecting logs from multiple sources such as applications, web servers, databases, and networks.
2. **Preprocessing:** Cleaning, filtering, and structuring logs for analysis.
3. **Storage and Management:** Storing logs using distributed platforms such as Hadoop or cloud storage systems.
4. **Analysis and Visualization:** Applying statistical analysis, machine learning, and visualization tools to extract insights.
5. **Real-Time Monitoring:** Using stream processing frameworks like Flink for continuous monitoring and alerts.

Research Work

Large-scale data analytics has become essential because modern systems continuously generate massive volumes of heterogeneous data, including logs from networks, servers, applications, industrial devices, and healthcare platforms. Processing such datasets efficiently requires big data technologies that balance **speed, accuracy, scalability, and cost-effectiveness**. However, designing systems that are simultaneously **large-scale, adaptive, fault-tolerant, accurate, and error-free** remains challenging due to the complexity of distributed environments and the unpredictable nature of real-world data streams [1], [2]. As a result, a wide range of big data tools and frameworks have emerged to address these requirements, including **Hadoop, MapReduce, HPCC, Storm, Qubole, Cassandra, CouchDB, Pentaho, Flink, and Cloudera**, among others [1]. These tools support advanced processing capabilities such as large-scale storage, distributed computation, machine learning integration, and real-time analytics across diverse applications [3], [4].

Among these technologies, **Apache Hadoop** has been widely adopted as a foundational framework for big data analysis because it enables data processing directly where the data is stored, minimizing transfer overhead and improving efficiency [2], [5]. Hadoop's architecture combines the **Hadoop Distributed File System (HDFS)** for distributed storage and **MapReduce** for parallel batch processing, enabling organizations to store large volumes of structured and unstructured log data across clusters and analyze them efficiently [2], [6]. HDFS provides a reliable storage layer by splitting files into blocks and replicating them across nodes, ensuring scalability and fault tolerance in large deployments [7]. At the same time, MapReduce supports batch-oriented processing by dividing computation into map and reduce tasks, allowing log aggregation, historical trend identification, and pattern extraction such as security threats or error frequency analysis [8], [9]. In addition, Hadoop

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

ecosystems often integrate tools such as **Hive** and **Pig**, enabling SQL-like queries and script-based analysis for extracting insights from stored log datasets [10], [11]. Despite its strengths, Hadoop-based log analysis typically suffers from **high latency** and is less suited for real-time monitoring due to the batch processing nature and operational complexity of setup and maintenance [12], [13].

A major challenge within Hadoop environments is **job scheduling**, particularly for MapReduce tasks where resource allocation, deadline requirements, data locality, and execution time significantly influence performance [14]. Research in Hadoop scheduling highlights that MapReduce schedulers can be grouped into categories such as **resource-aware, deadline-aware, data-locality-aware, learning-aware, budget-aware, and makespan-aware**, each designed to optimize specific performance objectives [1], [15]. These schedulers differ in their goals, algorithms, and evaluation approaches, and their effectiveness depends heavily on workload characteristics and cluster configurations [16], [17]. This demonstrates that efficient scheduling is a key factor in improving Hadoop-based log analytics performance, especially under heavy data loads [18].

In this research work, three major big data technologies are considered for log analysis: **Hadoop, Spark, and Flink**. Hadoop is primarily useful for **storage and batch analytics**, where the goal is to process large historical log datasets cost-effectively [2], [19]. For example, Hadoop supports log storage, aggregation, and pattern mining across months or years of data, making it well suited for compliance auditing and historical incident analysis [20]. However, when organizations require faster insights, Hadoop's limitations in real-time processing become more visible [12], [21].

To overcome latency issues, **Apache Spark** is often used because of its **in-memory computation engine**, which accelerates log processing by keeping intermediate results in memory rather than repeatedly reading from disk [22]. Spark supports both batch and near real-time analytics through components such as **Spark Core**, **Spark Streaming**, **Spark SQL**, and **MLlib**, enabling use cases such as fraud detection, network security monitoring, log pattern mining, and anomaly detection using machine learning [22], [23]. Spark provides unified analytics and broad API support (Python, Scala, Java, SQL), but typically requires higher memory resources and introduces additional operational complexity compared to Hadoop [22], [23].

Finally, **Apache Flink** is a powerful distributed processing framework designed for **stateful computations** over both bounded (batch) and unbounded (streaming) data streams. Flink supports **unified stream and batch processing**, advanced state management, low-latency processing, and high fault tolerance, making it especially effective for real-time log analytics such as monitoring, alerting, IoT log processing, and predictive maintenance [3], [4], [23]. Flink's event-time processing model enables accurate computation even when log events arrive out of order, which is common in distributed systems [4], [23]. Moreover, its scalable parallel execution and flexible APIs—including DataStream, DataSet, SQL, and Table APIs—allow organizations to build robust and efficient log analysis pipelines across multiple domains [3], [23].

Table 1 Comparative Analysis of Hadoop, Spark, and Flink for Log Analysis

Feature	Hadoop (MapReduce)	Apache Spark	Apache Flink
Primary Processing Model	Batch processing	Batch + Streaming (micro-batch)	Streaming-first + Batch
Latency	High (minutes)	Low (seconds; micro-batch)	Ultra-low (milliseconds–seconds)
Best Suited For	Historical log analysis, archival datasets	Fast analytics, near real-time monitoring	Real-time event processing, continuous monitoring

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Processing Speed	Slower (disk-based)	Very fast (in-memory execution)	Very fast (pipelined streaming + in-memory state)
Real-Time Log Support	No	Yes (micro-batching)	Yes (true streaming)
Handling Continuous Streams	Not supported	Supported via Spark Streaming	Built-in (native stream processing)
Fault Tolerance	Strong (data replication + re-execution)	Strong (RDD lineage recovery)	Strong (checkpointing + state recovery)
Stateful Processing	Limited	Supported (structured streaming)	Excellent (native stateful operators)
Event-Time Processing	Weak / Not natural	Moderate (structured streaming)	Excellent (native event-time + watermarks)
Scalability	Very high (large clusters)	High	High (optimized parallel execution)
Storage Support	HDFS mainly	HDFS, S3, HBase, others	HDFS, S3, Kafka, others
Ease of Setup	Complex	Easier than Hadoop	Moderate–Complex
Ease of Use (Programming)	Moderate–Hard	Easier (high-level APIs)	Moderate (requires stream concepts)
Programming APIs	Java, Pig, Hive	Scala, Python, Java, SQL	Java, Scala, SQL, Table API
Machine Learning Support	Limited	Strong (MLlib)	Growing (integration with ML pipelines)
Typical Log Use Cases	Batch log aggregation, offline reporting	Pattern mining, anomaly detection, interactive analytics	Fraud detection, anomaly detection, alerting, IoT monitoring
Resource Usage	Low-cost but slower	Higher memory required	Efficient but requires tuning for state
Cost Effectiveness	Best for cheap batch processing	Good for fast analytics (needs memory)	Best for real-time intelligence (needs tuning)
Main Limitation	High latency, no real-time	Streaming is micro-batch (not true real-time)	More complex to implement and manage

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

3. Aim and Objectives

The aim of this research is to develop a **scalable and efficient log analysis system** that can examine **multiple types of log files collected from worldwide sources** using Apache Flink. The proposed approach emphasizes real-time processing, parallel execution, and scalability, so that large volumes of log data can be analyzed effectively on standard computer hardware. Since modern systems generate diverse logs such as firewall logs, mail logs, server logs, cloud logs, and IoT logs, the system is designed to support heterogeneous formats and produce meaningful insights for monitoring, security, and operational decision-making.

The **primary objective** of this study is to design and implement an **intelligent log analysis framework** capable of processing diverse log formats while supporting both **real-time analytics and historical trend analysis**. To achieve this, the framework focuses on building an end-to-end pipeline that includes log collection, ingestion, processing, storage, visualization, and alerting. The system is intended to enable faster detection of anomalies and security threats, improve reliability through fault tolerance mechanisms, and generate actionable insights through graphical reporting.

The **specific objectives** include building an efficient log ingestion pipeline using tools such as Kafka, Fluentd, or Logstash, and designing a normalization layer that converts structured, semi-structured, and unstructured log data into a common schema. The framework will support **stream processing** using Flink (or Spark Streaming) for real-time monitoring and anomaly detection, while also enabling **batch processing** using Hadoop or Spark for historical analysis. To ensure scalability, distributed storage solutions such as HDFS, Amazon S3, and Elasticsearch will be integrated, along with indexing and query optimization methods for faster retrieval. In addition, the system aims to incorporate machine learning techniques for anomaly detection, fraud detection, and predictive analytics using tools such as Spark MLlib, TensorFlow, or Flink ML. Furthermore, visualization dashboards and automated alerting mechanisms will be implemented through Grafana, Kibana, or Superset to support real-time incident response.

Another key objective is to take full advantage of Apache Flink's strengths—particularly its **event-driven stream processing, stateful computation, checkpointing, fault tolerance, and scalability**—to improve the efficiency and reliability of log analysis. Finally, the study will benchmark Flink's performance against other frameworks such as Hadoop and Spark to evaluate its suitability for global-scale log analytics. Overall, the research aims to deliver a robust, fault-tolerant, and scalable system capable of processing massive log streams continuously and generating timely insights for different real-world domains.

4. Proposed Methodology

For this research endeavour, a variety of widely used log file formats collected from global sources were examined. The proposed approach requires that the **Apache Flink framework** first be used to evaluate and validate log files, after which their key contents are analyzed and presented through **graphical visualization**. This combination of real-time processing and graphical interpretation is highly valuable for supporting decision-making in **hospital management**, **industrial operations**, and other domains that rely on large-scale log data.

1. Significance of Using Apache Flink for Large Log Datasets: Apache Flink is a powerful stream-processing framework that is well suited for handling large and complex log datasets. Its **scalable, fault-tolerant, and real-time architecture** allows organizations to process continuous streams of log records with low latency and high throughput. As a result, Flink supports improved operational efficiency, faster detection of anomalies or security threats, and the extraction of actionable insights from large volumes of log data.

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

2. Role of Java in Enhancing Data Analysis Efficiency: Java improves the functionality and efficiency of log data analysis through several key features. First, it offers rich built-in data structures such as arrays, lists, sets, and maps, which support efficient data organization and processing. Second, Java provides strong multithreading capabilities, enabling parallel execution of tasks and faster handling of large datasets. Finally, Java integrates smoothly with major big data frameworks such as Apache Hadoop, Apache Spark, and Apache Flink, allowing developers to build robust and efficient analytics systems within these ecosystems.

3. Importance of Graphical Representation for Decision-Making: Graphical representation plays a crucial role in interpreting complex log data for effective decision-making in both hospitals and industries. Visual tools such as charts, graphs, and dashboards simplify large datasets and highlight trends, patterns, and anomalies. For example, doctors can use visual summaries to monitor patient-related events and device alerts, while industrial managers can identify system failures, production bottlenecks, and performance inefficiencies. Therefore, visualization enhances clarity, improves operational awareness, and supports timely data-driven decisions.

4. Examples of Practical Implementations: Several hospitals and industries have implemented log-based monitoring systems to improve management practices, such as real-time security auditing, patient device monitoring, predictive maintenance, and system performance tracking. These implementations demonstrate the effectiveness of combining large-scale log processing with visualization for operational improvement.

5. Future Developments in Log Data Analysis Technologies: Future improvements in log analysis technologies may include the integration of **machine learning and AI-based anomaly detection**, edge computing for faster processing at the data source, and more intelligent dashboards that provide predictive insights. These advancements can further enhance automation, scalability, and decision support in both industry and hospital environments.

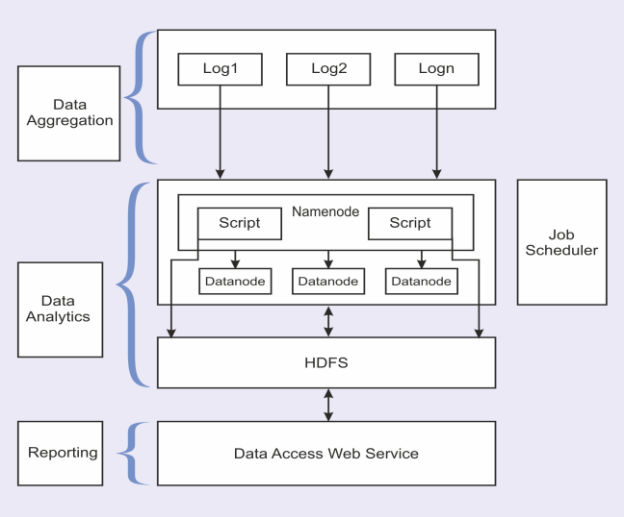

Figure 1. System Architecture of Hadoop and Map Reduce.

Step-Wise Explanation of Diagram 1: Hadoop-Based Log File Analysis Architecture

Diagram 1 illustrates a log file analysis system architecture designed using Hadoop-based technologies. The architecture processes large-scale logs collected from multiple sources and generates meaningful reports through distributed storage and computation. The major components are explained step by step as follows:

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Step 1: Data Aggregation (Log Collection)

- Log data is generated from various systems such as **Log1, Log2, …, Logn**.
- These logs are collected and ingested into the processing system for further analysis.
- The aggregation layer ensures that log data from multiple sources is available in one pipeline.

Step 2: Data Storage in HDFS

- After ingestion, all collected logs are stored in the **Hadoop Distributed File System (HDFS)**.
- HDFS uses a **NameNode** to manage metadata (file locations and structure) and multiple **DataNodes** to store log blocks.
- This distributed storage provides reliability, scalability, and fault tolerance for large datasets.

Step 3: Data Analytics (Distributed Processing)

- Once stored in HDFS, the log files are processed using distributed computing frameworks.
- Processing tasks are executed through scripts and analytics engines such as **MapReduce**, **Apache Spark**, or similar big data tools.
- These frameworks extract insights such as patterns, summaries, trends, and anomalies from log datasets.

Step 4: Job Scheduling and Workflow Automation

- A **Job Scheduler** is used to automate and manage the execution of log processing tasks.
- It schedules recurring jobs, monitors execution, and ensures workflow consistency.
- Tools such as **Apache Oozie**, **Apache Airflow**, or even **Cron jobs** may be used for scheduling.

Step 5: Reporting and Data Access

- After processing, the analytics results are made available through a **Data Access Web Service**.
- This service enables users to query processed log outputs and generate reports or dashboards.
- Reporting supports decision-making for system administrators, hospital management, and industrial operations.

Algorithm 1: Hadoop MapReduce–Based Global Log File Analysis (Batch Mode)

Input: Heterogeneous raw log files (Firewall, Web, Server, Mail, Call data)
Output: Aggregated summaries, frequency metrics, anomaly counts, stored analysis results in HDFS

Steps:

1: **Start**
2: Initialize Hadoop cluster services (NameNode, DataNode, ResourceManager, NodeManager)
3: Configure HDFS parameters (block size, replication factor, storage paths)
4: Collect log files from distributed sources: $Log_1, Log_2, \ldots, Log_n$
5: Create HDFS directories by log category (e.g., /logs/firewall/, /logs/web/, /logs/mail/)
6: Upload logs into HDFS using distributed storage commands

7: **Preprocessing Stage**
8: for each log file in HDFS do
9: Read log file line-by-line
10: Remove duplicates and empty records

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

11: Handle invalid timestamps and missing critical fields
12: Normalize each record into standard schema
13: Generate intermediate key-value format:
14: $Key \leftarrow$ (IP/User/EventType/TimeWindow)
15: $Value \leftarrow 1$ (or event metadata)
16: end for

17: **MapReduce Stage**
18: Define **Mapper** function
19: for each log record do
20: Parse required fields (timestamp, IP, event type, status)
21: Emit intermediate pair $(Key, 1)$
22: end for

23: Shuffle and sort intermediate outputs by key (Hadoop internal operation)

24: Define **Reducer** function
25: for each unique Keydo
26: Compute aggregation: $Total \leftarrow \sum Value$
27: Emit final output pair $(Key, Total)$
28: end for

29: Store reducer output to HDFS directory /output/log_summary/

30: **Post-Processing Stage (Optional)**
31: Execute additional MapReduce jobs for trends, anomalies, and top-N queries
32: Export results to Hive/HBase/ElasticSearch for reporting
33: Generate dashboards and summary reports
34: **End**

Algorithm 2: Apache Spark–Based Log File Analysis (In-Memory Batch + Streaming)

Input: Raw logs from HDFS/S3/Kafka (structured, semi-structured, unstructured)
Output: Fast log summaries, anomaly indicators, streaming alerts, stored metrics

Steps

1: **Start**
2: Initialize Spark environment (SparkSession, executors, memory, cluster mode)
3: Load input logs from storage source:
4: HDFS/S3 for batch mode, Kafka for streaming mode
5: Convert logs into Spark DataFrames/Datasets using schema inference or predefined schema

6: **Data Cleaning and Normalization Stage**
7: Remove corrupted entries, null timestamps, and duplicate records
8: Standardize timestamps into common time zone (UTC)
9: Parse log formats (JSON/CSV/XML/Syslog/Custom) and extract structured fields:
10: (timestamp, source, level, eventType, IP/user, message, status)

11: **Batch Analytics (Spark Core / Spark SQL)**
12: Perform distributed in-memory transformations: filter(), select(), groupBy(), agg()
13: Compute frequency-based summaries (Top IPs, Top Users, Error counts)
14: Compute time-window trends using window() aggregations
15: Store batch results into distributed output storage (HDFS/S3/HBase/ElasticSearch)

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

16: **Streaming Analytics (Optional: Structured Streaming)**
17: if real-time logs are available then
18: Read log stream from Kafka topics
19: Apply watermarking and sliding/tumbling windows
20: Compute continuous metrics and anomaly triggers
21: if anomaly condition is satisfied then
22: Generate alert record and push to alert channel/dashboard
23: end if
24: Write streaming outputs into sink (ElasticSearch/Cassandra/PostgreSQL)
25: end if

26: **Machine Learning Extension (Optional: Spark MLlib)**
27: Extract features (TF-IDF, categorical encoding, frequency vectors)
28: Train or apply anomaly detection / classification models
29: Store prediction results for visualization and reporting

30: Generate dashboards and reports using Kibana/Grafana/Power BI
31: **End**

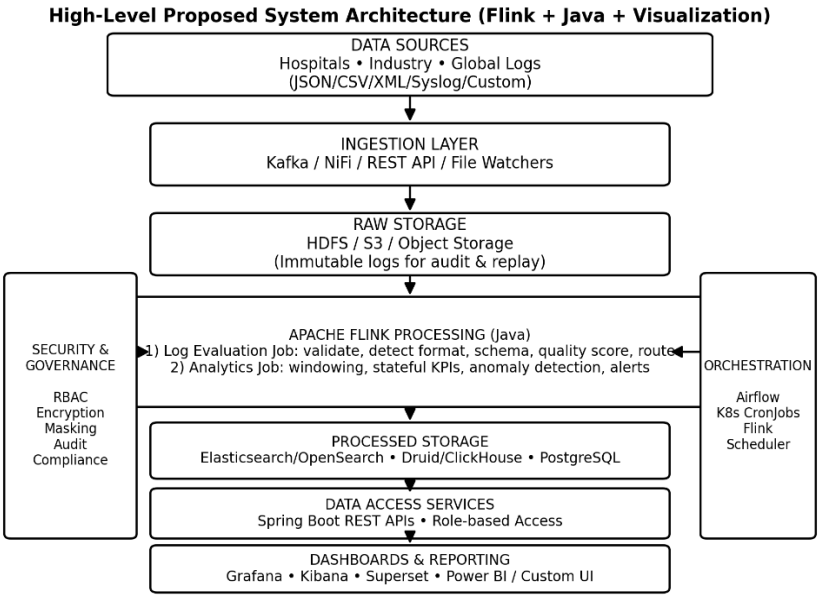

Figure 2. High-level proposed system architecture

Figure 2 illustrates a high-level proposed system architecture for large-scale log file evaluation and graphical analysis using Apache Flink and Java. In this architecture, log data generated from multiple sources such as hospitals, industrial systems, and global platforms (in formats like JSON, CSV, XML, Syslog, and custom logs) is first collected through an ingestion layer using Kafka, NiFi, REST APIs, or file watchers. The ingested logs are stored in raw form within distributed storage such as HDFS or S3 for audit and replay purposes. Apache Flink, implemented with Java, then performs two major tasks: (1) log evaluation to validate formats, detect schemas, calculate data quality, and route logs accordingly, and (2) analytics processing to compute window-based KPIs, identify anomalies, and generate alerts in real time. The processed outputs are stored in analytical databases such as Elasticsearch/OpenSearch, Druid/ClickHouse, and PostgreSQL, and then exposed through Spring Boot REST APIs with role-based access control. Finally, the results are presented through visualization platforms such as Grafana, Kibana, Superset, Power BI, or custom dashboards to support effective decision-making in hospital and industry management, while security and governance (RBAC, encryption, masking, auditing,

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

compliance) and orchestration tools (Airflow, Kubernetes CronJobs, Flink scheduler) ensure reliability, automation, and secure operations across the pipeline.

Algorithm 3: Flink-Based Log Evaluation and Graphical Analytics Pipeline

Input:
Raw log streams/files $L = \{l_1, l_2, \ldots, l_n\}$from hospitals, industries, and global systems

Output:
Validated logs L_v, analytics results R, anomaly alerts A, and graphical dashboards D

Steps

1: **Initialize system components**
2: Configure ingestion sources (Kafka / REST / File Watchers)
3: Configure raw storage (HDFS/S3) and processed storage (Elastic/Druid/PostgreSQL)
4: Deploy Flink cluster with state backend and checkpointing enabled

5: **Ingest Logs**
6: for each incoming log record $l_i \in L$do
7: Send $l_i$to ingestion layer
8: Store $l_i$in Raw Storage (HDFS/S3)
9: end for

10: **Log Evaluation Stage (Flink Job 1)**
11: for each log record $l_i$do
12: Detect log format f_i(JSON/CSV/XML/Syslog/Custom)
13: Extract schema $s_i$and mandatory fields
14: if $l_i$is invalid OR missing timestamp/critical fields then
15: Route $l_i$to Error Topic / Reject Store
16: else
17: Compute log quality score q_i
18: Enrich metadata (source, type, device, department, etc.)
19: Append validated log to L_v
20: end if
21: end for

22: **Log Analytics Stage (Flink Job 2)**
23: for each validated log $l_v \in L_v$do
24: Apply event-time alignment and watermarking
25: Perform window-based aggregation (time windows: min/hour/day)
26: Compute KPIs (failure rate, access counts, downtime, utilization, etc.)
27: Perform anomaly detection using thresholds/pattern rules
28: if anomaly detected then
29: Generate alert $a_j$and push to Alert Channel
30: Add $a_j$to A
31: end if
32: Store metrics and summaries into Processed Storage
33: end for

34: **Data Access and Visualization**
35: Expose processed results via Spring Boot REST APIs
36: Generate dashboards Dusing Grafana/Kibana/Superset/Power BI
37: Return dashboards, reports, and alerts to hospital/industry decision-makers

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

End Algorithm

5. Mathematical Equations for Log File Analysis

A. Firewall Log Analysis (Blocked IP Frequency and Source-IP Entropy)

In firewall log analysis, two common statistical measures are (i) **the frequency of blocked source IPs**, and (ii) **the entropy of source IP distribution**. The **blocked IP frequency** quantifies how often a particular IP address is blocked relative to the total number of block events, which helps identify suspicious or repeatedly offending sources [1]. Let Ndenote the total number of blocked events in the log, and let $n(\text{ip}_i)$represent the number of times source IP $\text{ip}_i$is blocked. Then, the frequency of a blocked IP is computed as:

$$f(\text{ip}_i) = \frac{n(\text{ip}_i)}{N} \qquad (1)$$

For example, if $n(\text{ip}_1) = 3$, $n(\text{ip}_2) = 1$, and $n(\text{ip}_3) = 6$, then $N = 3 + 1 + 6 = 10$, resulting in $f(\text{ip}_1) = 0.3$, $f(\text{ip}_2) = 0.1$, and $f(\text{ip}_3) = 0.6$. In addition, entropy is widely used to measure the **uncertainty or randomness** in the distribution of source IPs; low entropy suggests that traffic is concentrated from a few sources, whereas high entropy indicates widely distributed traffic, which may be associated with scanning or distributed attacks [2]. If $p(\text{ip}_i)$is the probability of observing IP ip_i, then the Shannon entropy of the source IP distribution is given by:

$$H = -\sum_{i=1}^{m} p(\text{ip}_i)\log_2 p(\text{ip}_i) \qquad (2)$$

where $m$is the number of distinct source IPs and $p(\text{ip}_i) = f(\text{ip}_i)$.

B. Mail Log Analysis (Sender Frequency and Bayesian Spam Detection)

For mail log analysis, a basic but effective metric is the **frequency of senders**, which indicates how active a particular sender is compared with overall email traffic [1]. Let Mbe the total number of emails and $m(s_i)$be the number of emails sent by sender s_i. Then sender frequency is defined as:

$$f(s_i) = \frac{m(s_i)}{M} \qquad (3)$$

For instance, if $M = 1000$and Sender A sends $m(A) = 50$, then $f(A) = 0.05(5\%)$, while if Sender B sends $m(B) = 200$, then $f(B) = 0.20(20\%)$. Beyond basic frequency measures, **Bayes' theorem** is widely applied in spam filtering to compute the probability that an email is spam given its observed content (e.g., words in the email) [3]. Using Bayes' rule, the posterior spam probability is computed as:

$$P(\text{spam} \mid x) = \frac{P(x \mid \text{spam})P(\text{spam})}{P(x)} \qquad (4)$$

where $x$denotes the observed email content, $P(\text{spam})$is the prior spam probability, $P(x \mid \text{spam})$is the likelihood of observing content $x$in spam, and $P(x)$is the marginal probability of observing content x. For classification, the marginal term can be expanded using the law of total probability [3]:

$$P(x) = P(x \mid \text{spam})P(\text{spam}) + P(x \mid \text{ham})P(\text{ham}) \qquad (5)$$

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

These probability-based models provide a mathematically grounded approach for detecting spam and identifying suspicious sender behavior in large-scale mail logs [3].

Algorithm 4: Log Evaluation and Statistical Analysis for Firewall and Mail Logs

Notation

- L_f: Firewall log dataset (block events)
- L_m: Mail log dataset (email events)
- N: Total number of blocked events in L_f
- M: Total number of emails in L_m
- $\mathcal{I}$: Set of distinct source IPs in L_f
- $\mathcal{S}$: Set of distinct senders in L_m
- $n(\text{ip})$: Count of blocked events for IP ip
- $m(s)$: Count of emails sent by sender s
- $f(\text{ip})$: Frequency of blocked IP ip
- $f(s)$: Frequency of sender s
- $p(\text{ip})$: Probability of observing ipin block events
- H: Shannon entropy of source IP distribution
- x: Observed email content features (e.g., word set)
- $P(\text{spam})$, $P(\text{ham})$: Prior probabilities
- $P(x \mid \text{spam})$, $P(x \mid \text{ham})$: Likelihoods
- $P(\text{spam} \mid x)$: Posterior probability of spam

Input

- Firewall logs L_f(block events with source IPs)
- Mail logs L_m(email events with sender + content features)
- Priors $P(\text{spam}), P(\text{ham})$and likelihood model parameters

Output

- Blocked-IP frequency map $\{f(\text{ip})\}$
- Source-IP entropy H
- Sender frequency map $\{f(s)\}$
- Spam classification probability $P(\text{spam} \mid x)$for each email

Steps

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

1: **Initialize counts and sets**
2: Set $N \leftarrow 0, M \leftarrow 0$
3: Initialize hash maps $n(\cdot)$for IP counts, and $m(\cdot)$for sender counts
4: Initialize $\mathcal{I} \leftarrow \emptyset, \mathcal{S} \leftarrow \emptyset$

A. Firewall Log Processing (Frequency + Entropy)

5: for each firewall event $e \in L_f$do
6: Extract source IP ip $\leftarrow e$.srcIP
7: $n(\text{ip}) \leftarrow n(\text{ip}) + 1$
8: $N \leftarrow N + 1$
9: $\mathcal{I} \leftarrow \mathcal{I} \cup \{\text{ip}\}$
10: end for

11: for each ip $\in \mathcal{I}$do
12: Compute blocked IP frequency:
13: $f(\text{ip}) = \frac{n(\text{ip})}{N}$
14: end for

15: Initialize entropy $H \leftarrow 0$
16: for each ip $\in \mathcal{I}$do
17: Compute probability $p(\text{ip}) \leftarrow f(\text{ip})$
18: Update entropy:
19: $H \leftarrow H - p(\text{ip})\log_2(p(\text{ip}))$
20: end for

B. Mail Log Processing (Sender Frequency + Bayesian Spam Probability)

21: for each mail event $r \in L_m$do
22: Extract sender $s \leftarrow r$.sender
23: Extract content features $x \leftarrow r$.features
24: $m(s) \leftarrow m(s) + 1$
25: $M \leftarrow M + 1$
26: $\mathcal{S} \leftarrow \mathcal{S} \cup \{s\}$

27: Compute sender frequency:
28: $f(s) = \frac{m(s)}{M}$

29: Compute spam posterior probability using Bayes:
30: $P(\text{spam} \mid x) = \frac{P(x|\text{spam})P(\text{spam})}{P(x|\text{spam})P(\text{spam})+P(x|\text{ham})P(\text{ham})}$

31: if $P(\text{spam} \mid x) \geq \tau$then
32: Label email as **Spam**
33: else
34: Label email as **Ham**
35: end if
36: end for

37: **Return** $\{f(\text{ip})\}, H, \{f(s)\}, P(\text{spam} \mid x)$

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

End Algorithm

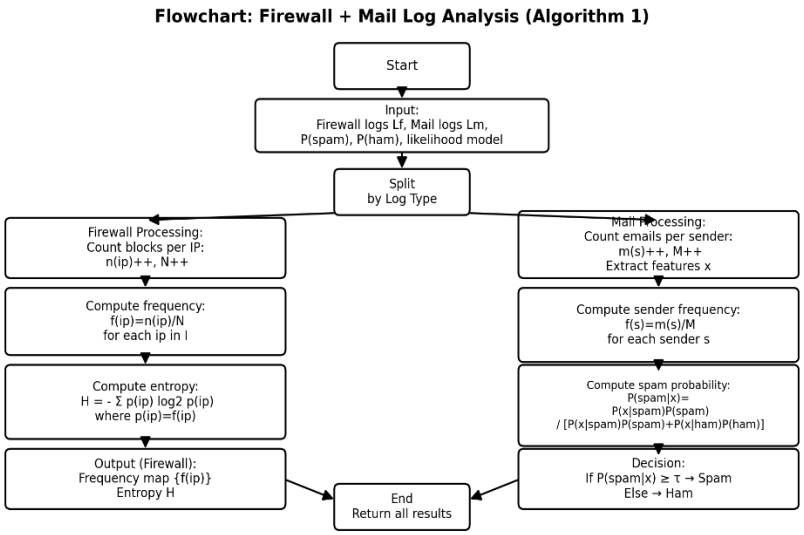

Figure 3. The complete processing workflow

Figure 3 represents the complete processing workflow for **Algorithm 1**, which performs mathematical analysis on both **firewall logs** and **mail logs** to support large-scale log monitoring and decision-making. The process begins by accepting inputs consisting of firewall log data L_f, mail log data L_m, and probability parameters such as $P(\text{spam})$, $P(\text{ham})$, and likelihood models for spam detection. The workflow then splits into two parallel branches based on log type: in the **firewall log branch**, the system counts the number of blocked events for each source IP $n(\text{ip})$and total blocked events N, computes the frequency of each blocked IP using $f(\text{ip}) = n(\text{ip})/N$, and then calculates the source-IP entropy $H = -\sum p(\text{ip})\log_2 p(\text{ip})$to quantify randomness in the distribution of blocked IPs. In the **mail log branch**, the system counts emails per sender $m(s)$and total emails M, calculates sender frequency using $f(s) = m(s)/M$, and estimates the probability that an email is spam through Bayes' theorem $P(\text{spam} \mid x)$, followed by a decision step that classifies each email as spam or ham based on a threshold τ. Finally, the flowchart ends by returning all computed results, including blocked-IP frequency maps, entropy values, sender frequencies, and spam classification outputs for further reporting and dashboard visualization.

6. Experimental Work

Single Cluster Node (Pseudo-Distributed Mode) Setup

I. Terminology

When configuring Hadoop on a single machine, it is important to use correct terminology. A **single-node (pseudo-distributed) cluster** means that all Hadoop services (daemons) run on one physical system, but they behave as if they are operating in a distributed environment. This mode uses **HDFS** and allows users to test Hadoop features such as NameNode, DataNode, and MapReduce locally.

In contrast, **Standalone mode** refers to Hadoop's default configuration where Hadoop runs as a single Java process without HDFS. Standalone mode is mainly used for debugging and basic testing and does not represent a distributed setup. Therefore, for a realistic local cluster simulation, pseudo-distributed mode is the appropriate choice.

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

II. Creating a Dedicated Hadoop System User

Although it is not mandatory, it is strongly recommended to create a dedicated user account specifically for running Hadoop. This improves system organization and enhances security because Hadoop files and processes remain separate from other applications and users. It also simplifies permission management, backups, and troubleshooting.

To create a Hadoop group and a dedicated Hadoop user (for example, hduser), the following commands can be used:

sudo addgroup hadoop

sudo adduser --ingroup hadoop hduser

These commands create:

- a group named **hadoop**
- a user named **hduser** assigned to the **hadoop** group

III. Configuring SSH Access

Hadoop requires **SSH access** to manage nodes, even when running on a single-node cluster. In pseudo-distributed mode, Hadoop daemons communicate through SSH to the local system. Therefore, SSH access to **localhost** must be configured for the hduser account.

Step 1: Generate an SSH Key

Login as hduser and generate an RSA key pair without a password:

su - hduser

ssh-keygen -t rsa -P ""

An empty passphrase is used here so that Hadoop can automatically access SSH without asking for user input each time. Although passwordless keys are not recommended for normal secure systems, they are commonly used for Hadoop cluster operations.

Step 2: Enable SSH Login to Localhost

Copy the public key into the authorized keys list:

cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys

ssh localhost

When connecting the first time, the system will ask:

Are you sure you want to continue connecting (yes/no)?

Type **yes** to allow the connection.

IV. Disabling IPv6

On Ubuntu systems, Hadoop may bind to IPv6 addresses when network configurations use values like 0.0.0.0. If the system is not connected to an IPv6 network, enabling IPv6 provides no benefit and can cause networking problems in Hadoop. Therefore, disabling IPv6 is recommended for smooth Hadoop operation.

To disable IPv6, open the file:

sudo nano /etc/sysctl.conf

Then add the following lines at the end:

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

net.ipv6.conf.all.disable_ipv6 = 1

net.ipv6.conf.default.disable_ipv6 = 1

net.ipv6.conf.lo.disable_ipv6 = 1

To confirm whether IPv6 is disabled:

cat /proc/sys/net/ipv6/conf/all/disable_ipv6

- Output **0** → IPv6 enabled
- Output **1** → IPv6 disabled

V. Hadoop Installation

Download Hadoop from the official Apache mirror and extract it into a suitable directory. A common installation path is:

/usr/local/hadoop

After extraction, change the ownership of Hadoop files so the dedicated Hadoop user can manage them:

sudo chown -R hduser:hadoop /usr/local/hadoop

This ensures that all Hadoop directories and files are accessible to the hduser account and the hadoop group.

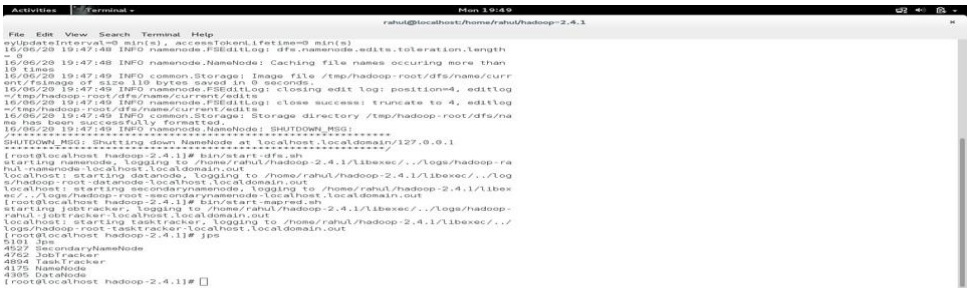

Figure 4. Running Single Cluster Node

To begin the distributed setup, **two single-node Hadoop clusters** must first be configured and successfully executed. In this arrangement, the Hadoop configuration is modified so that **one machine is designated as the Master node** (which also functions as a Slave node for local processing), while the **second machine is configured as a Slave node**. This initial setup ensures that both systems are operational before expanding to a fully distributed cluster environment.

For firewall log analysis, an application titled **"Applet Viewer: main1.class"** is used to process and visualize the log data, as summarized in the corresponding output image. The application provides a clear overview of network activity by presenting key traffic statistics. In the **Top 5 Hosts** section, the internal IP addresses that generated the highest number of connection attempts (HITS) are displayed. Among these, host **192.168.0.6** recorded the highest activity with **112 hits**, followed by host **192.168.0.7** with **56 hits**. The **Day-Wise Analysis** table presents the distribution of traffic hits across specific dates in **November 2023**, where the highest traffic was observed on **November 20** and **November 22**, each recording **64 hits**. Additionally, the **Action State** chart illustrates the proportion of firewall decisions, showing how traffic was handled by the firewall. The results indicate that **12.3%** of traffic was marked as **OPEN** (allowed), **27.3%** was recorded as **CLOSE** (actively denied, typically with a reset response), and the majority, **60.3%**, was classified as **DROP**, meaning the packets were silently blocked and discarded without any notification to the source.

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

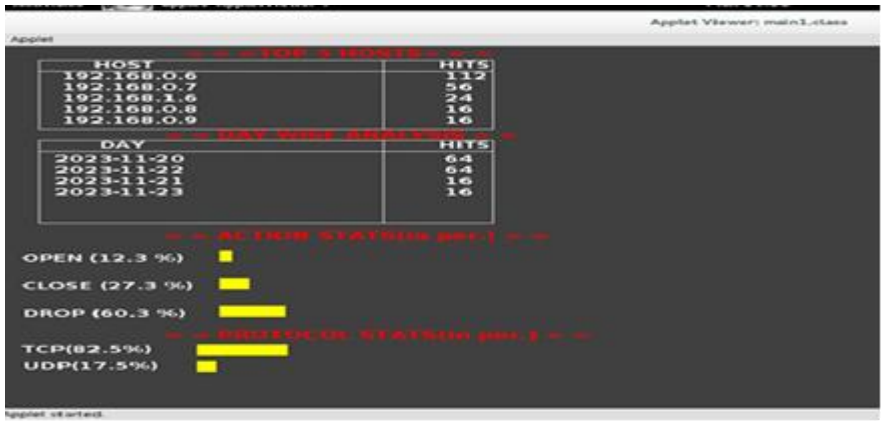

Figure 5. Firewall log analysis

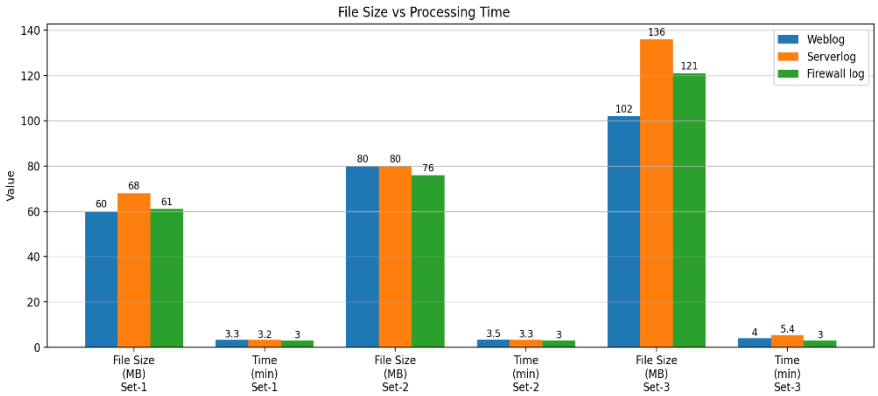

Figure 6. Propose System analysis log file in time

Figure 6 illustrates the relationship between log file size (MB) and processing time (minutes) for three different log types weblog, serverlog, and firewall log across three experimental datasets (Set-1 to Set-3). The results show a clear increase in processing time as file size grows, indicating higher computational demand for larger datasets. In Set-1, the file sizes are 60 MB (weblog), 68 MB (serverlog), and 61 MB (firewall log), with processing times of 3.3, 3.2, and 3 minutes respectively. In Set-2, the sizes increase to 80 MB (weblog), 80 MB (serverlog), and 76 MB (firewall log), with corresponding times of 3.5, 3.3, and 3 minutes. The highest workload occurs in Set-3, where file sizes rise to 102 MB (weblog), 136 MB (serverlog), and 121 MB (firewall log), resulting in processing times of 4, 5.4, and 3 minutes respectively. Overall, the figure highlights that server logs require the most processing time at larger file sizes, while firewall logs remain comparatively faster, demonstrating differences in computational complexity across log types.

7. Conclusion

Log file analysis is a critical component of modern IT systems because it enables organizations to monitor performance, detect security threats, and support data-driven decision-making. Based on the proposed algorithms, firewall logs were effectively analyzed by computing blocked IP frequency and entropy, which helped identify repeated offenders and traffic randomness. Similarly, mail logs were processed using sender frequency and Bayesian spam probability to classify suspicious emails accurately. The results show that as log file size increases, processing time also grows, with server logs requiring the highest time in large datasets, while firewall logs remained comparatively faster. These findings highlight the importance of scalable processing frameworks. Real-time technologies such as

Research Article

Apache Flink and Spark provide low-latency insights, while Hadoop remains valuable for batch analysis of historical logs. Future work should focus on automation, intelligent anomaly detection, and enhanced scalability.

References

[1] Soudabeh Hedayati, Neda Maleki, Tobias Olsson, Fredrik Ahlgren, Mahdi Seyednezhad and Kamal Berahmand, " MapReduce scheduling algorithms in Hadoop: a systematic study ",Springer open 2023, https://doi.org/10.1186/s13677-023-00520-9.

[2] Kumar Rahul, Rohitash Kumar Banyal and Neeraj Arora," A systematic review on big data applications and scope for industrial processing and healthcare sectors", Springer open 2023, https://doi.org/10.1186/s40537-023-00808-2.

[3] K Bapayya Naidu a , , B. Ravi Prasad Mohammed Saleh Al Ansari e , Samar Mansour Hassen , R. Vinod , M. Nivetha , Chamandeep Kaur , B. Kiran Bala," Analysis of Hadoop log file in an environment for dynamic detection of threats using machine learning", ELSEVIER 2022.

[4] Tang Jingwen, Chen Weimin, Zhang Min, Peng Kaikang, Lai Chaohao, Fu Kai ," Research on log data analysis technology based on improved Hadoop", IEEE 2022

[5] K Bapayya Naidu, B. Ravi Prasad, Mohammed Saleh Al Ansari , Samar Mansour Hassen , R. Vinod f , M. Nivetha, Chamandeep Kaur , B. Kiran Bala," Analysis of Hadoop log file in an environment for dynamic detection of threats using machine learning", ELSEVIER 2022.

[6] Hatim Talal Almansouri, "Hadoop Distributed file system for big data analysis", IEEE 2019.

[7] Shintaro YAMAMOTO, Shinsuke MATSUMOTO, Sachio SAIKI, Masahide NAKAMURA, "Materialized View as a Service for Large-Scale House Log in Smart City", 2013 IEEE, DOI 10.1109/CloudCom.2013.154

[8] Wulun Du, Depei Qian , Ming Xie, Wei Chen, "Research and Implementation of MapReduce Programming

Oriented GraphicalModeling System" , 2013 IEEE.

[9] Rahul Khullar, Tushar Sharama, "Addressing challenges hadoop for big data analysis" , 978-1-5386-2459-3/18, IEEE 2018.

[10] Yingxun Fu, Shilin Wen, and Li Ma," Data Adaptively Storing Approach for Hadoop Distributed File System", IEEE 2017

[11] Vaishali Sontakke, Dr. Dayanand R B," *Optimization of Hadoop MapReduce Model in cloud Computing Environment*" , IEEE Xplore Part Number: CFP19P17-ART; ISBN:978-1-7281-2119-2.

[13] Hemant Hingave ," *An approach for MapReduce based Log analysis using Hadoop*" , I CECS '2015 ,IEEE.

[14] Bina Kotiyal, Ankit Kumar, Bhaskar Pant, RH Goudar ," Big Data: Mining of Log File through Hadoop", IEEE 2018

[15] Xijiang Ke, Rai Jin, Xia Xie, Jie Cao ," A Distributed SVM Method based on the Iterative MapReduce", IEEE ICSC 2015, February 7-9, 2015

[16] Yandong Wang Huansong Fu Weikuan Yu, "Cracking Down MapReduce Failure Amplification through Analytics Logging and Migration" , 2015 IEEE.

[17] Meisuchi Naisuty, Achmad Nizar Hidayanto, Nabila Clydea Harahap, Ahmad Rosyiq, Agus Suhanto, and George Michael Samuel Hartono, "Data protection on hadoop distributed file system by using encryption algorithums: a systamatic literature review", doi:10.1088/1742-6596/1444/1/012012

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

[18] Paolo Bonacquisto, Giuseppe Di Modica," A procurement auction market to trade residual Cloud computing capacity" , DOI 10.1109/TCC.2014.2369435, IEEE Transactions on Cloud Computing

[19] Dazhao Cheng, Jia Rao, Yanfei Guo, Changjun Jiang and Xiaobo Zhou ," Improving Performance of HeterogeneousMapReduce Clusters with Adaptive Task Tuning" , DOI 10.1109/TPDS.2016.2594765, IEEE

[20] Kala Karun. A, Chitharanjan. K ," A Review on Hadoop – HDFS Infrastructure Extensions" , 2013 IEEE Conference on Information and Communication Technologies (ICT 2013), 978-1-4673-5758-6/13.

[21] Sun-Yuan Hsieh , Senior Member, IEEE, Chi-Ting Chen, Chi-Hao Chen, Tzu-Hsiang Yen, Hung-Chang Hsiao, and Rajkumar Buyya ," Novel Scheduling Algorithms for Efficient Deployment of MapReduce Applications in Heterogeneous Computing Environments" , IEEE TRANSACTIONS ON CLOUD COMPUTING, VOL. 6, NO. 4, OCTOBER-DECEMBER 2018

[22] Mamta Mittal, D. Jude Hemanth, Valentina Emilia Balas and Ragavendra Kumar (eds), " Big Data for Parallel Computing" in the Advances in Parallel Computing Series,IOS Press, pp 14, 2018

[23] Dazhao Cheng, Jia Rao, Yanfei Guo, Changjun Jiang and Xiaobo Zhou ," Improving Performance of HeterogeneousMapReduce Clusters with Adaptive Task Tuning" , DOI 10.1109/TPDS.2016.2594765, IEEE